International Journal of Sciences: Basic and Applied Research (IJSBAR)

ISSN 2307-4531
(Print & Online)

http://gssrr.org/index.php?journal=JournalOfBasicAndApplied

The DevOps Goes to the Embedded System

Acep Taryana[a*], Ari Fadli[b]

[a,b]*Dept of Electrical Engineering, Engineering Faculty, Universitas Jenderal Soedirman*

[a]*Email: aetthea71@gmail.com*

[b]*Email: fadli.te.unsoed@gmail.com*

Abstract

Lately, DevOps has been widely discussed in various papers, in addition the industry has implemented it to be a solution for developing and distributing software for general purposes. However, the application of DevOps to embedded systems is still quite difficult, and there is still lack of paper to discuss it. The purpose of this study is to explore the DevOps approach in developing embedded systems. The method used in this research is to use the Object-Oriented Software Engineering approach. Various tools used to support the development of embedded systems include Visual Paradigm, Quantum Modeling. The results explain that DevOps can be applied in the development of embeded systems through Forward and Reverse Engineering. Forward engineering includes analysis, design of the class, design of the state machine, design of coding, generate code, uploading some codes into board. While reverse engineering is reverse from forward engineering. However, the applying of DevOps is not still one stop services. It is signed by the displacement of tools when doing state-machine design, there is a shift in the use of tools, from the visual paradigm to quantum modeling.

Keywords: Software Engineering; SDLC; Development and Operation; Object-oriented technology.

1. Information System versus Embedded System

Information systems relating to the development of applications that are general, do not discuss the specific hardware development. Examples of information systems development are applications for hospitals, applications for education. While the embedded system is a discussion of developing systems related to hardware. Examples of embedded system development are vending machine development, elevator development.

----

* Corresponding author.

What's interesting about both systems development models, we begin by reviewing software development methods. In general, software development uses various paradigms or approaches such as waterfall, agile, and others. Whatever the approach, software development through the stages of requirements, analysis, design, programming, testing, deployment [1]. Then, there is a question, whether the stages of software development can be applied in the development of embedded systems? Embedded systems developed through software development methods are very appropriate [2,3]. Information system software is implemented on general-purpose computers, whereas software for embedded systems, there are pieces of software that will be installed in specific hardware environments for specific purposes, and are usually installed on various boards such as Arduino UNO [4], Texas Instrument [4], Texas Instrument [4]. Furthermore, what distinguishes the two is during the implementation of testing. Testing for general information systems is quite carried out in a computing environment whereas for testing embedded systems requires testing software that has been implanted in hardware that operates in a real environment [5]. It is very complicated to carry out software testing in a real environment for embedded system requirements [6]. However mathematical science approaches must be applied in the need to develop embedded systems that are very stricky (hard) [2]. However in the discussion of this paper, what is seen is the similarity of the two. To what extent do the two have the same pattern of activities that can be portrayed by a software engineering? The extent to which software development engineering can be applied by both. How each stage is applied in the development activities of both? Do all phases including requirements, analysis, design, coding, and testing can be used in the development phase.

1.1 DevOps

DevOps is currently a very hot topic to be discussed among practitioners and academics in software development. DevOps is a practical approach to software development supported by various tools to integrate the work of developers with the operations team [7]. There are academics who discuss DevOps as a Project Management approach [8], and there are also peeling about how DevOps as a continuation of the practical Agile method [9]. DevOps is a practical method that can be used to develop software for general purposes such as information systems development [10], and also devops can be used to develop special needs software such as embedded systems [11,6,3]. Both of these objects, we view as a system that is no difference. How these system objects are placed in the DevOps cycle. Please note that DevOps has a development and operation cycle [12,13,1]. Development includes Plan, Code, Build, Testing, and Release. While the operation phase includes Deploy, Operate, and Monitor. In fact, DevOps has been adopted to develop various software for specific purposes. For example, software development for university quality assurance systems [10]. In this paper, it has been discussed how the process of developing a quality assurance system software can be carried out by integrating the two institutions which include institutions that handle quality assurance, and institutions that handle software development. Quality assurance institutions as customers, while software development institutions as developers. The paper also shows how the collaboration path between the two. Another example is how DevOps tried to be applied in the development of embedded systems [11,3]. This paper discusses how the potential of DevOps is applied and describes the stages of DevOps in an Embedded System environment. In this paper, we will discuss the steps to implement DevOps in software development to support the implementation of Embedded Systems. The contribution of the paper to science is the discussion of the development of embedded systems using object-oriented technology with the DevOps approach. The second part discusses object-oriented methods in UML notation as a tool for system development since analysis, design, and coding. And also discussed about how the results of a coding uploaded on a board for embedded systems.

2. Materials and Method

2.1 Materials

In this study we use the Visual Paradigm (VP) tool for system analysis and design. Then, the Quantum Modeling (QM) 5.0.1 tool, for designing artifacts that is very close to the development of embedded systems [14]. Computing development using a Mac computer with the following specifications, 1.6 GHz Dual-Core Intel Core I5 Processor, 8GB Memory 2133 MHz LPDDR3.

2.2 Method

In the academic world that Object-oriented Technology has been packaged in the concept of Object-oriented Software Engineering (OOSE) [15]. OOSE includes the level of object-oriented analysis (OOA), object-oriented design (OOD), and object-oriented programming (OOP). Each level has different technical tools, such as for OOA and OOD using VP, for OOP development using Eclipse tools, Netbeans. All of these tools are provided to facilitate the development of object-oriented technology. Then how to implement all the above stages into an embedded system environment. We use Quantum Modeling tools that have functions to create state-machine design, coding and implementation [14]. Coding that is ready to be uploaded into a limited device such as Arduino UNO. The stages in the first paragraph above are the Forward engineering stages, what about the Reverse engineering stage. Based on the literature that Reverse Engineering is reverse from Forward Engineering [16,17]. Therefore, reverse engineering is done by evaluating software in embedded systems. Software evaluation is carried out by reading the code, then the code is turned into class diagrams, and class diagrams are interpreted in analysis artifacts such as use cases.

2.3 Proposed System

An example system was developed for the Blinky Project. Where there is 1 lamp mounted. The behavior of the system is that the lamp turns on for a few seconds and then dies for a few seconds. In this discussion we use the example of the Blinky Project, which in this example only makes a state-machine from the Blinky Project and implements it in the Arduino UNO Board [14]. As our job is to add the OOA and OOD stages before the State-Machine is obtained. To do this we use Quantum Modeling tool version 5.0.1 which has the ability to design State-machine diagrams, design coding using C ++, generate code that has been designed according to the Framework, test, and upload to the Arduino UNO Board. Figure 1 shows the DevOps cycle where the cycle has development (Dev) and Operations (Ops) sections. The stages of OOA and OOD are done in the "Plan" section, program development is done in the "Coding" section, software testing is carried out in the "Testing" section and then so on is the "release" and "deploy" section for software deployment into the operating environment. In this case, the operating environment is to use Arduino UNO.

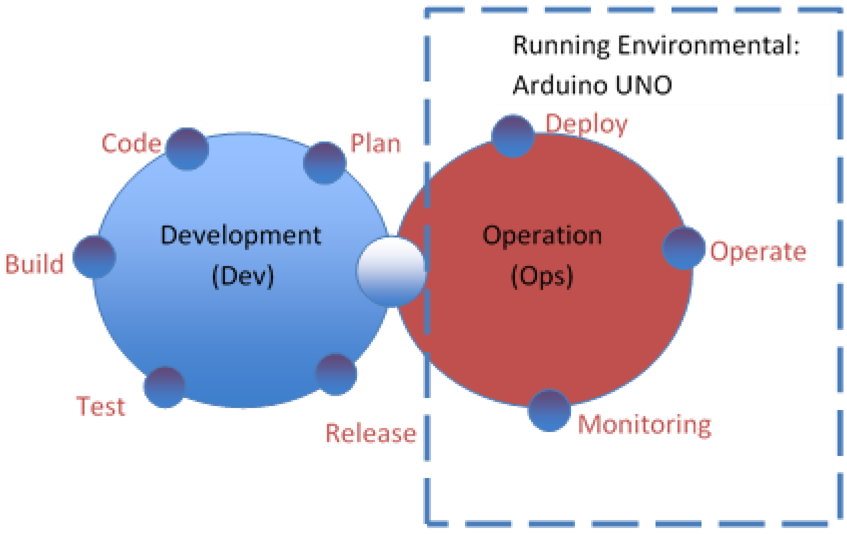


Figure 1: Proposed System for Opening the DevOps Concept in an Embedded System

3. Results

3.1 Analysis

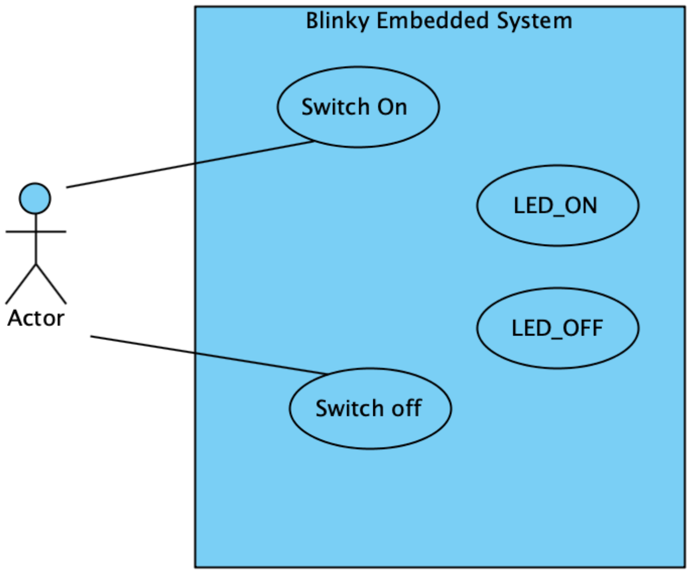


Figure 2: Use Case of a Blinky Embedded System

First, we must define a requirement for the system to be built. A requirement for the system to be developed is written as follows: Built an embedded system that has software and hardware components. The hardware component consists of an Arduino UNO device that is connected to an LED light. LED lights have a behavior of a few seconds and turn off a few seconds according to the specified time parameters. While the software component is written using Quantum Framework. Based on the statement of requirements in the first paragraph, we write the scope of the embedded system as in Figure 2. Figure 2 describes the relationship between the system and the external environment, the interaction of the system user actors with the system functions. The user actor has a duty to shut down the system or turn on the system. When the system is turned on by the user, the embedded system will function like a statement of requirements. The lights will experience 2 state, which are on or off for a few seconds.

3.2 Design of Classes

The second stage in developing this system is structuring information structuring into a class diagram concept. Based on Figure 1, we describe a class that can represent the scope of a blinky embedded system and the recording of information on LED lights in Figure 2. The class diagram has one class named "BSP". BSP stands for board support package. **BSP** is a software layer that has specific and routine hardware drivers that allow certain operating systems that have the ability to function in a hardware environment that is integrated with the operating system itself. The BSP class has a method of turning on the lights, called ledOn (), and to turn off the lights, it is named ledOff (). The attribute it has is ticks_per_second, it functions as a time parameter as the life span or death of the LED lights.

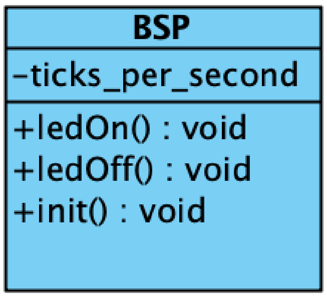


Figure 2: Blinky Class Diagram

3.3 Design of State-Machine

Furthermore, embedded systems are reactive systems that need to be described using the diagrams in UML. UML provides a state diagram that can explain the behavior of the system when the system is alive. Figure 3 is the state diagram. State diagram is also known as state-machine. The state-machine is described as coming from a Blinky Class Diagram. There are 2 states for LED lights namely state off and state on. The state off is passed at the start of the system or when the LED lights on after 60 seconds, while the on state is passed if the LED has been lit for more than 60 seconds.

Figure 3 is the final step in designing a Blinky Embedded System using a VP tool. Next, we must move to the QM tool to start designing the system using Quantum Framework. In the QM tool, Figure 3 is redrawn by adjusting the tool standard. Although different from Figure 3, the machine-state in Figure 4 still explains the behavior of the Embedded Blinky System.

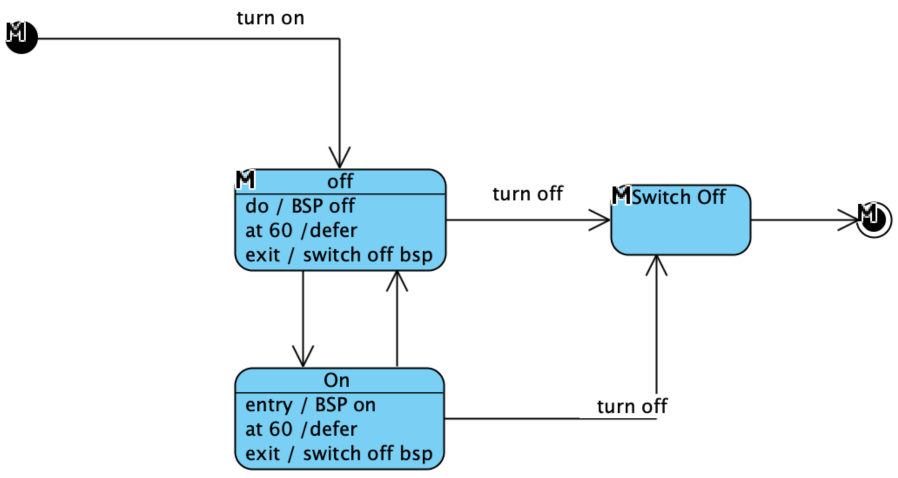


Figure 3: Blinky State-Machine Diagram

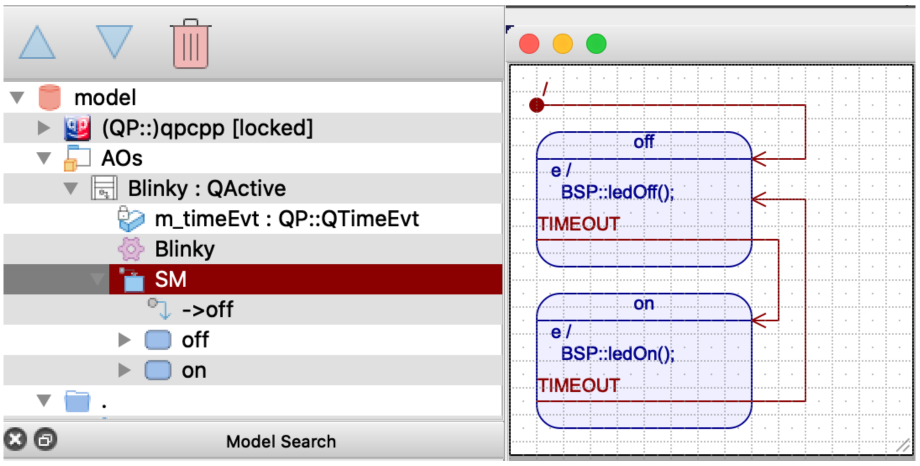
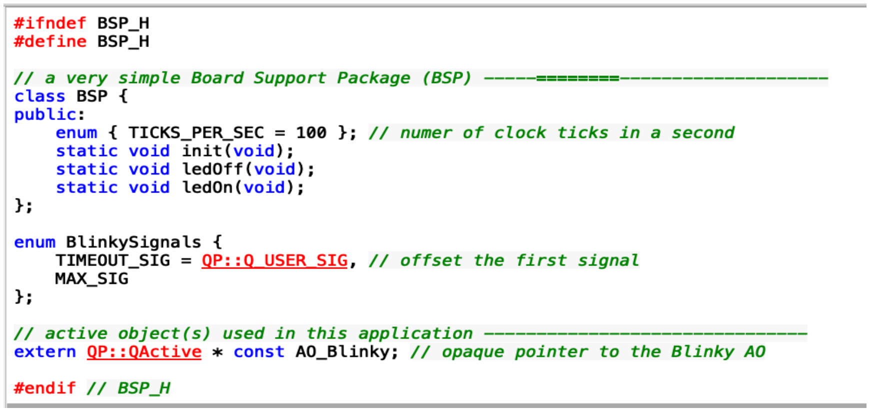


Figure 4: Blinky State-Machine in Quantum Modeling Tool

3.4 Coding

The third step is to draft the source code. There are 3 categories of source code that must be compiled, namely the header category as outlined in Source Code 1, the main program categories as outlined in Source Code 2, and the BSP program category as outlined in Source Code 3. Source Code 1 is written based on the State-machine in Figure 4. Because the source code is written in the Quantum Framework, all methods, attributes, parameters are bound by the rules of the referred framework

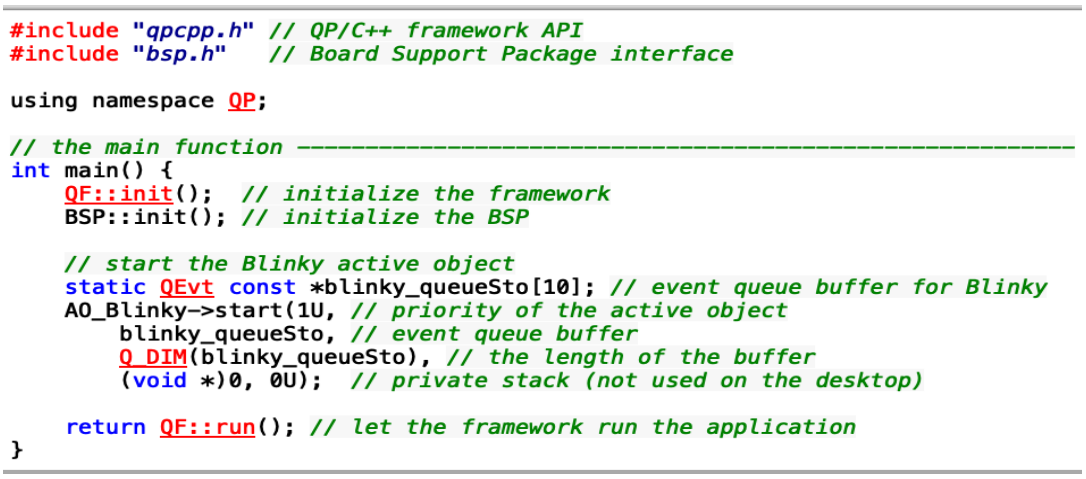
Source Code 1: Header of Blinky Project



```
#ifndef BSP_H
#define BSP_H

// a very simple Board Support Package (BSP) -----========--------------------
class BSP {
public:
    enum { TICKS_PER_SEC = 100 }; // numer of clock ticks in a second
    static void init(void);
    static void ledOff(void);
    static void ledOn(void);
};

enum BlinkySignals {
    TIMEOUT_SIG = QP::Q_USER_SIG, // offset the first signal
    MAX_SIG
};

// active object(s) used in this application -------------------------------
extern QP::QActive * const AO_Blinky; // opaque pointer to the Blinky AO

#endif // BSP_H
```

Figure 1

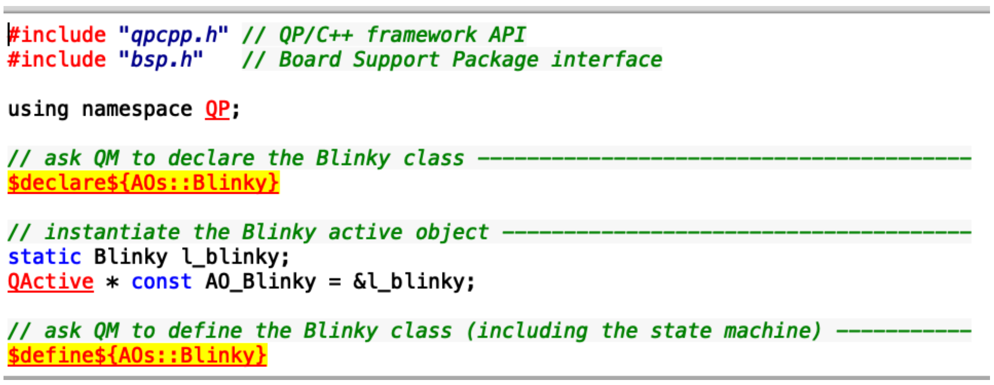
Source Code 2: Main Program



```
#include "qpcpp.h" // QP/C++ framework API
#include "bsp.h"   // Board Support Package interface

using namespace QP;

// the main function ---------------------------------------------------------
int main() {
    QF::init();  // initialize the framework
    BSP::init(); // initialize the BSP

    // start the Blinky active object
    static QEvt const *blinky_queueSto[10]; // event queue buffer for Blinky
    AO_Blinky->start(1U, // priority of the active object
        blinky_queueSto, // event queue buffer
        Q_DIM(blinky_queueSto), // the length of the buffer
        (void *)0, 0U);  // private stack (not used on the desktop)

    return QF::run(); // let the framework run the application
}
```

Figure 2

Source Code 3: Blinky Program BSP

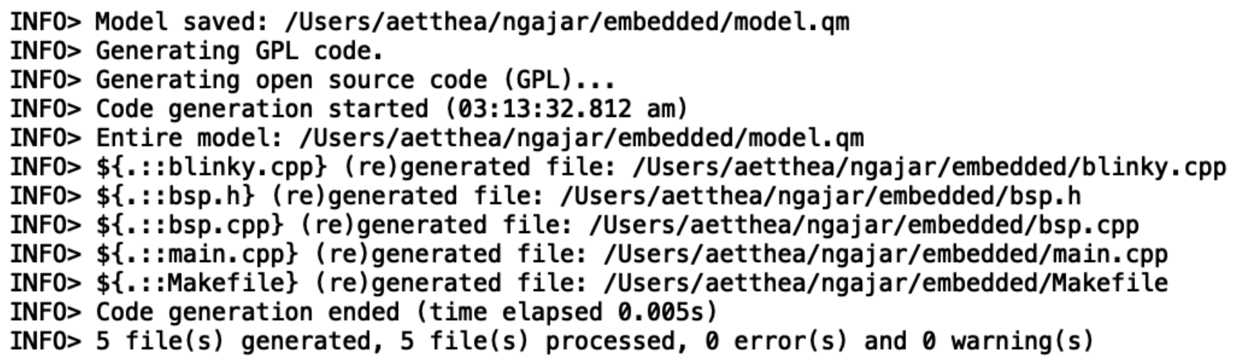


```
#include "qpcpp.h" // QP/C++ framework API
#include "bsp.h"   // Board Support Package interface

using namespace QP;

// ask QM to declare the Blinky class ----------------------------------------
$declare${AOs::Blinky}

// instantiate the Blinky active object --------------------------------------
static Blinky l_blinky;
QActive * const AO_Blinky = &l_blinky;

// ask QM to define the Blinky class (including the state machine) -----------
$define${AOs::Blinky}
```

Figure 3

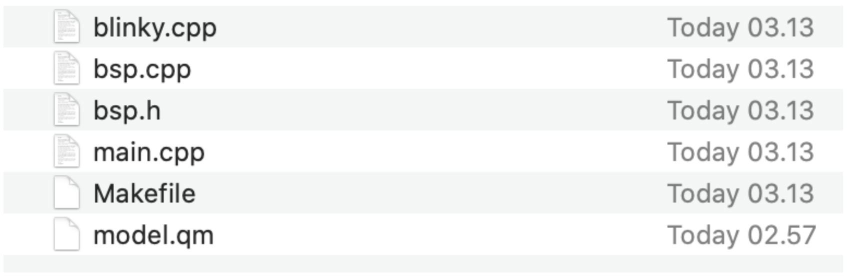
3.5 Generate Code

The fourth step is to generate source code from the design of Source Code 1, Source Code 2, and Source Code 3. Through QM tools we can generate source code according to the choice of languages in the framework. Generating produces several files that are ready to be uploaded into the intended BSP device such as Arduino UNO.



```
INFO> Model saved: /Users/aetthea/ngajar/embedded/model.qm
INFO> Generating GPL code.
INFO> Generating open source code (GPL)...
INFO> Code generation started (03:13:32.812 am)
INFO> Entire model: /Users/aetthea/ngajar/embedded/model.qm
INFO> ${.::blinky.cpp} (re)generated file: /Users/aetthea/ngajar/embedded/blinky.cpp
INFO> ${.::bsp.h} (re)generated file: /Users/aetthea/ngajar/embedded/bsp.h
INFO> ${.::bsp.cpp} (re)generated file: /Users/aetthea/ngajar/embedded/bsp.cpp
INFO> ${.::main.cpp} (re)generated file: /Users/aetthea/ngajar/embedded/main.cpp
INFO> ${.::Makefile} (re)generated file: /Users/aetthea/ngajar/embedded/Makefile
INFO> Code generation ended (time elapsed 0.005s)
INFO> 5 file(s) generated, 5 file(s) processed, 0 error(s) and 0 warning(s)
```

Output



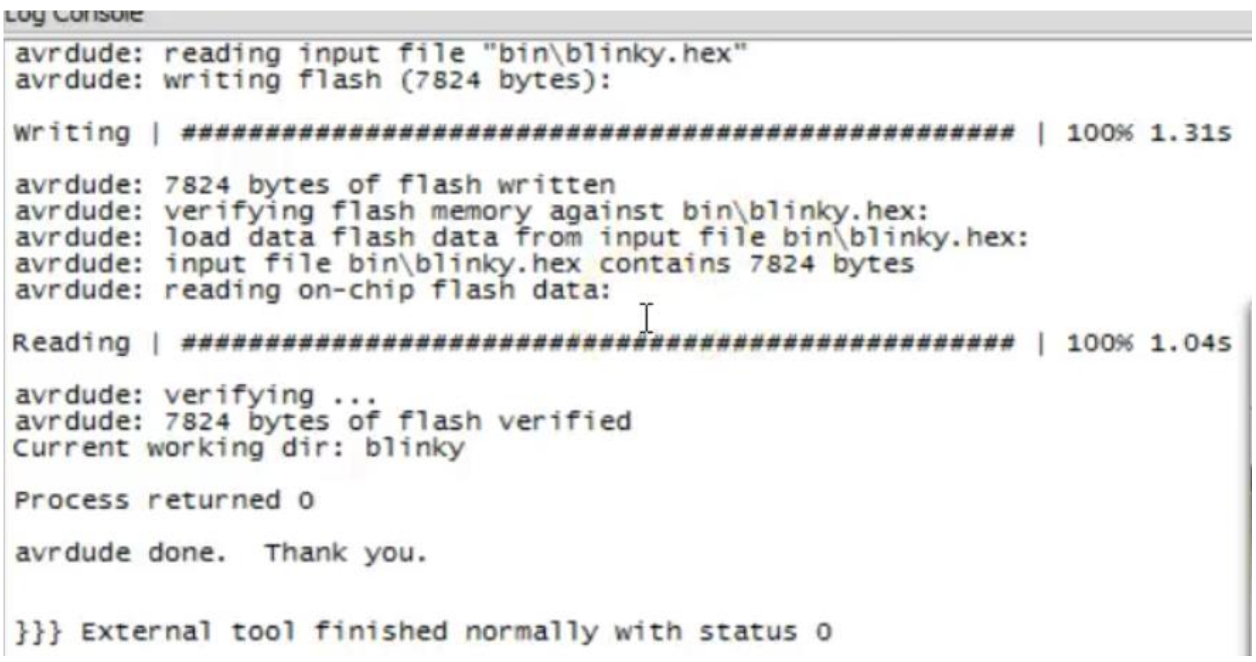
```
blinky.cpp      Today 03.13
bsp.cpp         Today 03.13
bsp.h           Today 03.13
main.cpp        Today 03.13
Makefile        Today 03.13
model.qm        Today 02.57
```

Figure 4

The generated code is ready to be tested in a computing environment using QM and if the test is successful then the code is ready to be uploaded into the intended Arduino UNO device.

3.6 Upload into Board

The final step is a trial to insert the Blinky source code into Arduino UNO. Figure 5 shows the process of uploading source code from QM to the Arduino UNO board. The uploading process has a successful status in accordance with the comments in Figure 5 below. With the status of "External tool finished normally with status 0" then the creation of the Blinky Project has been completed.



```
Log Console
avrdude: reading input file "bin\blinky.hex"
avrdude: writing flash (7824 bytes):

Writing | ################################################## | 100% 1.31s

avrdude: 7824 bytes of flash written
avrdude: verifying flash memory against bin\blinky.hex:
avrdude: load data flash data from input file bin\blinky.hex:
avrdude: input file bin\blinky.hex contains 7824 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 1.04s

avrdude: verifying ...
avrdude: 7824 bytes of flash verified
Current working dir: blinky

Process returned 0

avrdude done.  Thank you.


}}} External tool finished normally with status 0
```

Figure 5: The process of uploading the Source Code into Arduino UNO

3.7 Reverse Engineering

The steps above are Forward Engineering steps, since the analysis - design - coding - upload. In the Forward Engineering stage, the development from one stage to another experiences the use of tools that were from VP into QM when compiling the source code. This is done because the source code written must adjust to the framework used. What about the Reverse Engineering Blinky Project step? Of course reverse engineering can be carried out with a track of coding - design - analysis evaluations. Reverse Engineering development is the same as Forward Engineering, it cannot be done smoothly because of the change of tools between the design and coding stages. The work cannot be done in one piece of equipment, we call the work cannot be done with one stop service.

3.8 DevOps Approach

The stages of development have been discussed both in Forward and Reverse engineering. These stages are the two directions of development in the DevOps approach. Furthermore, like what DevOps is implemented in the Blinky Project. In DevOps, Dev is a development work that has a 0.5 part of Development and Ops has 0.5 of Development. The implementation of Forward and Reverse engineering in the development of Blinky Project has done 50% more work. The rest are Deploy, Monitoring and evaluation. In this paper only discussed the part that 50% development, the rest is future work.

4. Conclusion

Development of embedded systems using the DevOps approach is still not standard using a framework that is sustainable from one stage to another. The use of development tools will experience a shift or switch from one tool to another. In the analysis and design phase we can use VP but when we are going to implement coding, we have to move to other tools using QM. Unlike the development of non-embedded software systems, developers will work as a one-stop service throughout the entire development cycle. In addition, as a result of the development framework that is still not standardized, the process of reverse engineering the product is experiencing difficulties, because it has to move from one tool to another in the development of its software. The stages of advanced development consist of the stages of Analysis, Design of Classes, Design of State-Machine, Coding, Generate of Code, and Uploading. When creating state-machines we have to move to QM so that the design results in UML can be integrated with artifacts, libraries in the Quantum Framework. While the reverse development stage is reverse from advanced development.

References

[1]. A. Robert, T ; Masters, William ;Stark, “Teaching Agile Development with DevOps in a Software Engineering and Database Technologies Practicum,” in 3rd International Conference on Higher Education Advances, 2017, pp. 1353–1362.

[2]. M. Darbari and H. Ahmed, “Software Engineering Practices in Embedded System Design Using Discrete Modeling Techniques,” Proc. World Congr. Eng. 2010. London, U.K, vol. I, 2010.

[3]. B. Graaf, M. Lormans, and H. Toetenel, “Embedded Software Engineering: The State of the Practice,” IEEE Software, vol. 20, no. 6. pp. 61–69, Nov-2003.

[4]. Louis Leo, “WORKING PRINCIPLE OF ARDUINO AND USING IT AS A TOOL FOR STUDY AND RESEARCH,” Int. J. Control. Autom. Commun. Syst., vol. 1, no. 2, 2018.

[5]. L. Hatton, “Embedded software testing,” 2010.

[6]. L. E. Lwakatare et al., “Towards DevOps in the embedded systems domain: Why is it so hard?,” in Proceedings of the Annual Hawaii International Conference on System Sciences, 2016, vol. 2016-March, pp. 5437–5446.

[7]. F. M. A. Erich, C. Amrit, and M. Daneva, “A qualitative study of DevOps usage in practice,” in Journal of Software: Evolution and Process, 2017, vol. 29, no. 6.

[8]. L. BANICA, M. RADULESCU, D. ROSCA, and A. HAGIU, “Is DevOps another Project Management Methodology?,” Inform. Econ., vol. 21, no. 3/2017, pp. 39–51, 2017.

[9]. P. Raj and P. Sinha, “Project Management In Era Of Agile And Devops Methdlogies,” Int. J. Sci. Technol. Res., vol. 9, p. 1, 2020.

[10]. A. Taryana, I. Setiawan, A. Fadli, and E. Murdyantoro, “Pioneering the automation of lnternal quality assurance system of higher education (IQAS-HE) using DevOps approach,” in Proceedings - 2017 International Conference on Sustainable Information Engineering and Technology, SIET 2017, 2017, pp. 259–264.

[11]. P. E. Wijaya, I. Rosyadi, and A. Taryana, “An attempt to adopt DevOps on embedded system development: Empirical evidence,” in Journal of Physics: Conference Series, 2019, vol. 1367, no. 1.

[12]. J. F. Pérez, W. Wang, and G. Casale, “Towards a DevOps approach for software quality engineering,” in WOSP-C 2015 - Proceedings of the 2015 ACM/SPEC Workshop on Challenges in Performance Methods for Software Development, in Conjunction with ICPE 2015, 2015, pp. 5–10.

[13]. J. Wettinger, U. Breitenbücher, and F. Leymann, “DevOpSlang - Bridging the gap between

development and operations," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 8745 LNCS, pp. 108–122, 2014.

[14]. "Modern Embedded Software," 2020. [Online]. Available: www.state-machine.com. [Accessed: 01-Jun-2020].

[15]. F. SoleimanianGharehchopogh, S. Jodati Gourabi, and I. Maleki, "Object Oriented Software Engineering Models in Software Industry," Int. J. Comput. Appl., vol. 95, no. 3, pp. 13–16, 2014.

[16]. I. D. Baxter and M. Mehlich, "Reverse engineering is reverse forward engineering," Sci. Comput. Program., vol. 36, no. 2, pp. 131–147, 2000.

[17]. C. Baidada, E. M. Bouziane, and A. Jakimi, "An Analysis and New Methodology for Reverse Engineering of UML Behavioral," Int. J. Adv. Eng. Manag. Sci., vol. 2, no. 7, pp. 1012–1016, 2016.